low level TeX

lines

Contents

1 Introduction 1
2 Warning 2
3 Constructing paragraphs 2
4 Subpasses 6
5 Definitions 11
6 Tracing 13
7 Criterion 15
8 Examples 16
9 Pages 17
10 Profiles 20

1 Introduction

There is no doubt that TEX does an amazing job of “breaking paragraphs into lines” where a paragraph is a sequence of words in the input separated by spaces or its equivalents (single line endings turned space). The best descriptions of how that is done can be found in Don Knuths “The TEX Book”, “TEX The Program” and “Digital Typography”. Reading and rereading the relevant portions of those texts is a good exercise in humility.

That said, whatever follows here builds upon what Knuth gave us and in no way we pretend to do better than that. It started out as a side track of improving rendering math in combination with more control over breaking inline math. It pretty much about having fun with the par builder but in the end can also help make your results look better. This is especially true for proze.

Trying to describe the inner working of the par builder makes no sense. Not only is it kind of complex, riddled with magic constants and heuristics, but there is a good chance for us to talk nonsense thanks to misunderstanding. However, some curious aspects will be brought up. Consider what follows a somewhat naive approach and whatever goes wrong, blame the authors, not TEX.

If you're one of those reader who love to complain about the bad manuals, you can stop reading here. There is plenty said in the mentioned books but you can also consult Viktor Eijkhouts excellent “TEX by Topic” (just search the web for how the get these books). If you're curious and in for some adventure, keep reading.

2 Warning

This is a first version. What is described here will stay but is still experimental and how it evolves also depends on what demands we get from the users. We have defined some experimental setups in ConTEXt. We wil try to improve the explanations in ways that (we hope) makes clear what happens deep down but that takes time. These might change depending on feedback. We assume that we're in granular mode:

```
\setupalign[granular]
```

We will explain below what that means, but let us already now make clear that this will likely become the default! As far as we can see, due to the larger solution space, the inter-word spacing is more even but that also means that some paragraphs can become one line less or more.

3 Constructing paragraphs

There are several concepts at work when TEX breaks a paragraph into lines. Here we assume that we talk about text: words separated by spaces. We also assume that the text starts at the left edge and nicely runs till the right edge, with the exception of the last line.

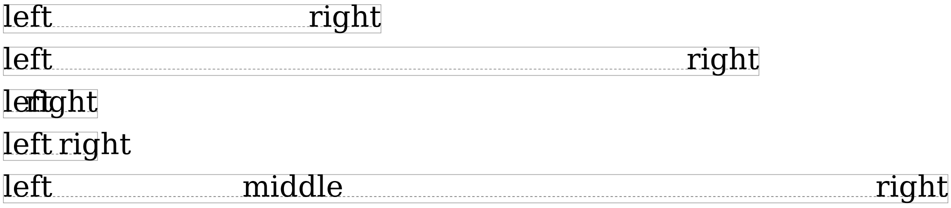
- The spaces between words can stretch or shrink. We don't want that to be too inconsistent (visible) between two lines. This is where the terms loose and tight come into play.
- Words can be hyphenated but we don't want that to happen too often. We also discourage neighboring lines to have hyphens. Hyphenating the (pre) final line is also sort of bad.
- We definitely don't want words to stick out in the margin. If we have to choose, stretching is preferred over shrinking. If spaces become too small words, start to blur.
- If needed glyphs can stretch or shrink a little in order to get rid of excessive spacing. But we really want to keep it minimal, and avoid it when possible. Usually we permit more stretch than shrink. Not all scripts (and fonts for that matter) might work well with this feature.
- As a last resort we can stretch spaces so that we get rid of any still sticking out word. When TEX reports an overfull box (often a line) you have to pay attention!

When TEX decides where to break and when to finish doing so it uses a system of penalties and demerits and at some point makes decisions with regards to how bad a breakpoint (and eventually a paragraph) is. The penalties are normally relatively small unless we really want to penalize. When TEX is in the process of breaking a paragraph it calculates badness values for each line. This can be seen as a measure on how bad looking a line is; a badness of zero is good, but the larger the badness becomes, the worse the line is.

Here we shortly summarize the parameters that play a role in calculating what TEX calls the costs of breaking a line at some point: it's a combination of weighting penalties as well as over- or undershooting the line with, where the amount (dimension) and kind of (fillers) stretch and shrink determien the final verdict.

```
\ruledhbox to 20 ts{left \hss right}
\ruledhbox to 40 ts{left \hss right}
\ruledhbox to  5 ts{left \hss right}
\ruledhbox to  5 ts{left      right}
\ruledhbox to  5 es{%
    left
    \hskip 1ts plus 0.5ts\relax
    middle
    \hskip 1ts plus 1.5ts\relax
    right%
}
```

These boxes show a bit what happens with spacing that can stretch of shrink. The first three cases are not bad because it's what we ask for with the wildcard `\hss`.[1]



TEX will run over each paragraph at most three times. On each such run, it will choose different breakpoints, calculate badness of each possible line, combine that with eventual penalties, and calculate a certain demerit value for each possible paragraph. It creats a set of solutions as it progresses, discards the worse cases so far and eventually ends of what it thinks is best.

---

[1] We use this opportunity to promote the new `ts` and `es` units.

The process is primarily controlled by these parameters:

- `\pretolerance`: This number determines the success of the first, not hyphenated pass. Often the value is set to the plain TEX value of 100. If TEX finds a possible division of a paragraph such that no line has a badness higher than `\pretolerance`, the algorithm quits here and that line is chosen.

- `\tolerance`: This number determines the success of the second, hyphenated pass. Often the value is set to the plain TEX value of 200.

- `\emergencystretch`: This dimension kicks in when the second pass is not successful. In ConTEXt we often set it to `2\bodyfontsize`.

When we are (in ConTEXt speak) `tolerant`, we have a value of 3000, while `verytolerant` bumps it to 4500. These are pretty large values compared to the default 100 and 200 that seem to cover most cases well, especially when we have short words, a reasonable width and lots of opportunities for hyphenation. Keep in mind that a macro package has to default to values that make sense for the average case.

We now come to the other relevant parameters. You need to keep in mind that the demerits are made from penalty values that get squared which is why parameters with demerits in their name have high values: a penalty of 50 squared has to relate to a demerit of 5000, so we might have $2500 + 5000$ at some point.

The formula (most often) used to calculate the demerits d is

$$d = (l + b + p)^2 + e$$

Here l is the `\linepenalty`, set to 10 in plain, b is the badness of the line, and p is the penalty of the current break (for example, added by hyphenation, or by breaking an inline formula). The e stands for extra non-local demerits, that do not depend on only the current line, like the `\doublehyphendemerits` that is added if two lines in a row are hyphenated.

The badness reflects how the natural linewidth relates to the target width and uses a cubic function. A badness of zero is of course optimal, but a badness of 99 is pretty bad. A magic threshold is 12 (around that value a line is considered decent). If you look at the formula above you can now understand why the line penalty defaults to the low value of 10.

- `\hyphenpenalty`: When a breakpoint occurs at a discretionary this one gets added. In LuaMetaTEX we store penalties in the discretionary nodes but user defined `\discretionary`'s can carry dedicated penalties. This value is set to 50, which is not that much. Large values reduce the solution space so best keep this one reasonable.

- `\linepenalty`: Normally this is set to 10 and it is the baseline for a breakpoint. This is again a small value compared to for instance the penalties that you find in inline math. There we need some breakpoints and after binary and relation symbols such an opportunity is created. The specific penalties are normally 500 and 700. One has to keep in mind, as shown in the formula above, that the penalties are not acting on a linear scale when the demerits are calculated. Math spacing and penalty control is discussed in the (upcoming) math manual.

- `\doublehyphendemerits`: Because it is considered bad to have two hyphens in a row this is often set pretty high, many thousands. These are treated as demerits (so outside of the squared part of the above formula).

- `\finalhyphendemerits`: The final (pre last) line having a hyphen is also considered bad. The last line is handled differently anyway, just because it gets normally flushed left.

- `\adjdemerits`: lines get rated in terms of being loose, decent, tight, etc. When two lines have a different rating we bump the total demerits.

- `\looseness`: it is possible to force less or more lines but to what extend this request is honored depends on for instance the possible (emergency) stretch in the spaces (or any glue for that matter). `

It is worth noticing that you can set `\lastlinefit` such that the spaces in the last line will be comparable to those in the preceding line. This is a feature that ε-TEX brought us. Anyways, keep in mind normally penalties are either small, or when we want to be tough, pretty high. Demerits are often relatively large.

The next one is a flag that triggers expansion (or compression) of glyphs to kick in. Those get added to the available stretch and/or shrink of a line:

- `\adjustspacing`: Its value determines if expansion kicks in: glyphs basically get a stretch and shrink value, something that helps filling our lines. We only have zero, two and three (and not the pdfTEX value of two): three means ‘only glyphs’ and two means ‘font kerns and glyphs’.

In LuaMetaTEX we also have:

- `\linebreakcriterion`: The normal distinction between loose, decent and tight in TEX uses 12 for 0.5 and 99 for about 1.0, but because we have more granularity (.25) we can set four values instead. The default of zero (`"0C0C0C63`) then becomes `"020C2A63`. When set that way the default `\adjdemerits` has to be halved 5000 so that we compare the more granular distances. Don't worry if you ‘don't get it’, hardly any user will change these values. One can think of the 100 squared becomes a 10000 (at least this helps relating these numbers) and 10000 is pretty bad in TEXs perception.

- `\adjustspacingstep`: When set this one is are used instead of the font bound value which permits local control without defining a new font instance.

- `\adjustspacingstretch`: idem.

- `\adjustspacingshrink`: idem.

- `\orphanpenalty`: This penalty will be injected before the last word of a paragraph.

- `\orphanpenalties`: Alternatively a series of penalties can be defined. This primitive expects a count followed by that number of penalties. These will be injected starting from the end.

The shape of a paragraph is determined by `\hangindent`, `\hangafter`, `\parshape` and `\parindent`. The width is controlled by `\hsize`, `\leftskip`, `\rightskip`. In addition there are `\parinitleftskip`, `\parinitrightskip`, `\parfillleftskip` and `\parfillrightskip` that control first and last lines.

We also have these:

- `\linebreakpasses`: When set to one, the currently set `\parpasses` will be applied.

- `\parpasses`: This primitive defined a set of sub passes that kick in when the second pass is finished. This basically opens up the par builder. It is still experimental and will be improved based upon user feedback. Although it is a side effect of improving the breaking of extensive mixes of math and text, it is also quite useful for text only (think novels).

In the next sections we will explain how these can improve the look and feel of what you typeset.

4 Subpasses

In LuaTEX and therefore also in LuaMetaTEX a paragraph is constructed in steps:

- The list of nodes that makes the paragraph is hyphenated: words become a mixture of glyphs and discretionaries.
- That list is processed by a font handler that can remove, add or change glyphs depending on how glyphs interact. This depends on the language and scripts used.
- The result is fed into the parbuilder that applies upto three passes as mentioned before.

In traditional TEX these three actions are combined into one and the overhead is shared. In the split case the processing time gets distributed and in practice the last action is not the one that takes most time. This is why the mechanism that we discuss next has little impact on the run: calling the par builder a few times more seldom results in more runtime. This is why in we support so called sub passes between the second and third one.

Here is an example of a setup. We set a low tolerance for the first pass and second pass. We can do that because we don't need to play safe nor need to compromise.

```
\pretolerance  75
\tolerance    150
\parpasses      3
    threshold           0.025pt
    classes             \indecentparpassclasses
    tolerance           150
  next
    threshold           0.025pt
    classes             \indecentparpassclasses
    tolerance           200
    emergencystretch     .25\bodyfontsize
  next
    threshold           0.025pt
    classes             \indecentparpassclasses
    tolerance           200
    optional            1
    emergencystretch     .5\bodyfontsize
\relax
\linebreakpasses 1
```

Because we want to retain performance we need to test efficiently if we really need the (here upto three) additional passes, so let's see how it is done. When a pass list is defined, and line break passes are enabled, the engine will check *after* the second pass if some more work is needed. For that it will do a quick analysis and calculate four values:

- overflow : the maximum value found, this is something really bad.
- underflow : the maximum value found, this is something we can live with.
- verdict : what is the worst badness of lines in this paragraph.
- classified : what classes are assigned to lines, think looseness, decent and tight.

There are two cases where the engine will continue with the applying passes: there is an overflow or there is a verdict (max badness) larger than zero. When we tested this on some large documents we noticed that this is nearly always true, but by checking we save a few unnecessary passes.

Next we test if a pass is really needed, and if not we check the next pass. When a pass is done, we pick up where we left, but we test for the overflow or badness every sub pass. The next checks make us run a pass:

- overfull exceeds threshold
- verdict exceeds badness
- classified overlaps classes

Here `threshold`, `badness` and `classes` are options in a pass section. Which test makes sense depends a bit on how TeX sees the result. Internally TeX uses numbers for its classification (0..5) but we map that onto a bitset because we want an overview:

		indecent	**almostdecent**	**loose**	**tight**
1	**veryloose**	+	+	+	
2	**loose**	+	+	+	
4	**semiloose**	+		+	
8	**decent**				
16	**semitight**	+			+
32	**tight**	+	+		+

The semiloose and semitight values are something LuaMetaTeX. In ConTeXt we have these four variants predefined as `\indecentparpassclasses` and such.

The sections in a par pass setup are separated by `next`. For testing purposes you can add `skip` and `quit`. The `threshold` tests against the overfull value, the `badness` against the verdict and `classes` checks for overlap with encountered classes, the classification.

You can specify an `identifier` in the first segment that then will be used in tracing but it is also passed to callbacks that relate to this feature. Discussing these callback is outside the scope fo this wrapup.

You need to keep in mind that parameters are not reset to their original values between two subpasses of a paragraph. We have `tolerance` and `emergencystretch` which are handy for simple setups. When we start with a small tolerance we often need to bump that one. The stretch is likely a last resort. The usual demerits can be set too: `doublehyphendemerits`, `finalhyphendemerits` and `adjdemerits`. We have `extrahyphenpenalty` that gets added to the penalty in a discretionary. You can also set `linepenalty` to a different value than it normally gets.

The `looseness` can be set but keep in mind that this only makes sense in very special cases. It's hard to be loose when there is not much stretch or shrink available. The `linebreakcriterion` parameter can best be left untouched and is mostly there for testing purposes.

The LuaMetaTEX specific `orphanpenalty` gets injected before the last word in a paragraph. High values can lead to overfull boxes but when used in text that hyphenate well or with languages that have short words it might work out well.

The next four parameters are related to expansion: `adjustspacing`, `adjustspacingstep`, `adjustspacingshrink` and `adjustspacingstretch`. Here we have several scenarios.

- Fonts are set up for expansion (in ConTEXt for instance with the quality specifier). When `hz` is then enabled it will always kick in.
- When we don't enable it, the par pass can do it by setting `adjustspacing` (to 3).
- When the other parameters are set these will overload the ones in the font, but used with the factors in there, so different characters get scaled differently. You can set the step to one to get more granular results.
- When expansion is *not* set on the font, setting the options in a pass will activate expansion but with the factors set to 1000. This means all characters are treated equal, which is less subtle.

When a font is not set up to use expansion, you can do something like this:

```
\parpasses    6
    classes              \indecentparpassclasses
    threshold            0.025pt
    tolerance             250
    extrahyphenpenalty     50
    orphanpenalty        5000
  % font driven
```

```
next ifadjustspacing
  threshold            0.025pt
  classes              \tightparpassclasses
  tolerance             300
  adjustspacing           3
  orphanpenalty        5000
next ifadjustspacing
  threshold            0.025pt
  tolerance            350
  adjustspacing           3
  adjustspacingstep       1
  adjustspacingshrink    20
  adjustspacingstretch   40
  orphanpenalty        5000
  emergencystretch     .25\bodyfontsize
% otherwise, factors 1000
next
  threshold            0.025pt
  classes              \tightparpassclasses
  tolerance             300
  adjustspacing           3
  adjustspacingstep       1
  adjustspacingshrink    10
  adjustspacingstretch   15
  orphanpenalty        5000
next
  threshold            0.025pt
  tolerance             350
  adjustspacing           3
  adjustspacingstep       1
  adjustspacingshrink    20
  adjustspacingstretch   40
  orphanpenalty        5000
  emergencystretch     .25\bodyfontsize
% whatever
next
  threshold            0.025pt
  tolerance            3000
  orphanpenalty        5000
  emergencystretch     .25\bodyfontsize
```

```
\relax
```

With `ifadjustspacing` you ignore steps that expect the font to be setup, so you don't waste time if that is not the case.

There is also a `callback` parameter but that one is experimental and used for special purposes and testing. We don't expect users to mess with that.

A really special feature is optional content. Here we use as example a quote from Digital Typography:

```
Many readers will skim over formulas on their first reading of your
exposition. Therefore, your sentences should flow smoothly when all but
the simplest formulas are replaced by \quotation {blah} or some other
\optionalword {1} {grunting }noise.
```

Here the `grunting` (with embedded space) is considered optional. When you set `\linebreakoptional` to 1 this word will be typeset. However, when you set the pass parameter `linebreakoptional` to 0 it will be skipped. There can be multiple optional words with different numbers. The numbers are actually bits in a bit set so plenty is possible. However, normally these two values are enough, if used at all.

5 Definitions

The description above is rather low level and in practice users will use a bit higher level interface. Also, in practice only a subset of the parameters makes sense in general usage. It is not that easy to decide on what parameter subset will work out well but it can be fun to play with variants. After all, this is also what TEX is about: look, feel and fun.

Some users praise the ability of recent TEX engines to provide expansion and protrusion. This feature is a bit demanding because not only does it add to runtime (although in LuaMetaTEX that normally can be neglected), it also makes the output files larger. Some find it hard to admit, but it even can result in bad looking documents when applied with extremes.

The traditional (MkIV) way to set up expansion is to add this to the top of the document, or at least before fonts get loaded.

and later on to enable it with:

```
\setupalign[hz]
```

However, par passes make it possible to be more selective. Take the following two definitions:

```
\startsetups align:pass:quality:1
    \pretolerance 50
    \tolerance    150
    \parpasses    6
        identifier           \parpassidentifier{quality:1}
        threshold            0.025pt
        tolerance            175
      next
        threshold            0.025pt
        tolerance            200
      next
        threshold            0.025pt
        tolerance            250
      next
        classes              \almostdecentparpassclasses
        tolerance            300
        emergencystretch     .25\bodyfontsize
      next ifadjustspacing
        classes              \indecentparpassclasses
        tolerance            300
        adjustspacing          3
        emergencystretch     .25\bodyfontsize
      next
        threshold            0.025pt
        tolerance            3000
        emergencystretch     2\bodyfontsize
    \relax
\stopsetups

\startsetups align:pass:quality:2
    \pretolerance 50
    \tolerance    150
    \parpasses    5
        identifier           \parpassidentifier{quality:2}
        threshold            0.025pt
        tolerance            175
      next
        threshold            0.025pt
        tolerance           200
      next
        threshold           0.025pt
        tolerance           250
      next ifadjustspacing
        classes             \indecentparpassclasses
        tolerance           300
        adjustspacing         3
        emergencystretch    .25\bodyfontsize
      next
        threshold           0.025pt
        tolerance           3000
        emergencystretch    2\bodyfontsize
    \relax
\stopsetups
```

You can now enable one of these:

```
\setupalignpass[quality:1]
```

The result is shown in figure 1 where you can see that expansion is applied selectively; you have to zoom in to see where.

6 Tracing

There are several ways to see what goes on. The engine has a tracing option that is set with `\tracingpasses`. Setting it to `1` reports the passes on the console, and a value of `2` also gives some details.

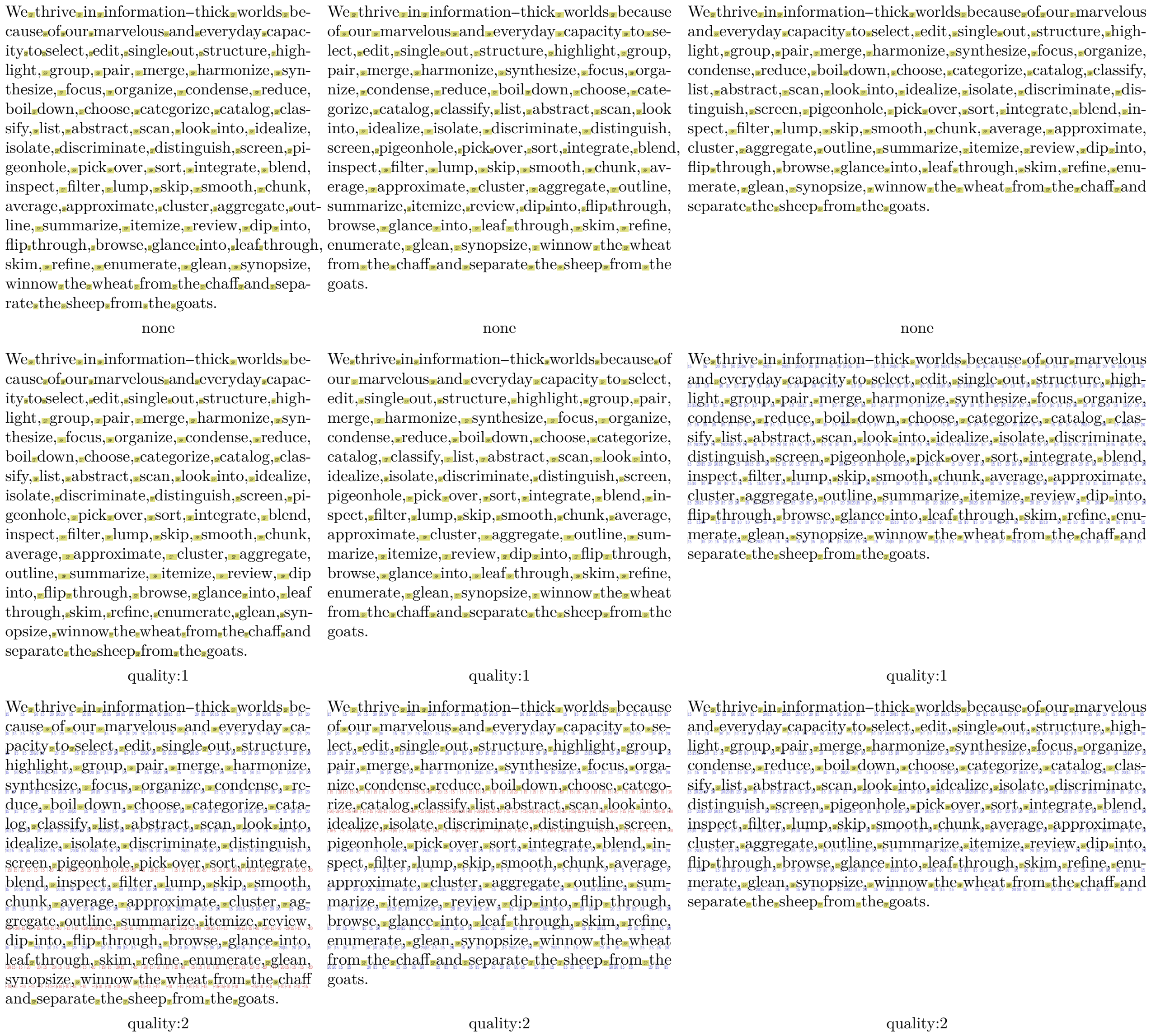
There is a also a tracker, `paragraphs.passes` that can be enabled. This gives a bit more information:

```
\enabletrackers[paragraphs.passes]
\enabletrackers[paragraphs.passes=summary]
\enabletrackers[paragraphs.passes=details]
```

If you want to see where expansion kicks in, you can use:

```
\showmakeup[expansion]
```

This is just one of the options, `spaces`, `penalties`, `glue` are are useful when you play with passes, but if you are really into the low level details, this is what you want:



We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

none

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

none

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

none

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

quality:1

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

quality:1

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

quality:1

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

quality:2

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

quality:2

We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

quality:2

Figure 1 Two different passes applied to `tufte.tex`.

```
\startnarrower[5*right]
\startshowbreakpoints[option=margin,offset=\dimexpr{.5\emwidth-\rightskip}]
\samplefile{tufte}
\stopshowbreakpoints
\stopnarrower
```

We thrive in information–thick worlds because of our marvelous and
everyday capacity to select, edit, single out, structure, highlight, group,
pair, merge, harmonize, synthesize, focus, organize, condense, reduce,
boil down, choose, categorize, catalog, classify, list, abstract, scan, look
into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick
over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk,
average, approximate, cluster, aggregate, outline, summarize, itemize,
review, dip into, flip through, browse, glance into, leaf through, skim, re-
fine, enumerate, glean, synopsize, winnow the wheat from the chaff and
separate the sheep from the goats.

```
[ 1] b=168 d=18.7196pt p=3.870 r=1.190 (dt=168) (0pt)
[ 2] b=0 d=-0.3461pt p=-0.072 r=0.037 (dt=0) (0pt)
[ 3] b=4 d=5.34726pt p=1.105 r=0.350 (dt=4) (0pt)
[ 4] b=0 d=1.06113pt p=0.219 r=0.055 (dt=0) (0pt)
[ 5] b=6 d=-2.53214pt p=-0.523 r=0.388 (dt=6) (0pt)
[ 6] b=43 d=14.83798pt p=3.068 r=0.754 (dt=43) (0pt)
[ 7] b=9 d=5.91122pt p=1.222 r=0.451 (dt=9) (0pt)
[ 8] b=12 d=-5.60977pt p=-1.160 r=0.501 (dt=13) (0pt)
[ 9] b=0 d=-1.6835pt p=-0.348 r=0.157 (dt=0) (0pt)
[ 10] b=0 d=201.56406pt p=41.672 r=201.564 (dt=0) (0pt)
```

You can see the chosen solutions with

```
\showbreakpoints[n=1]
```

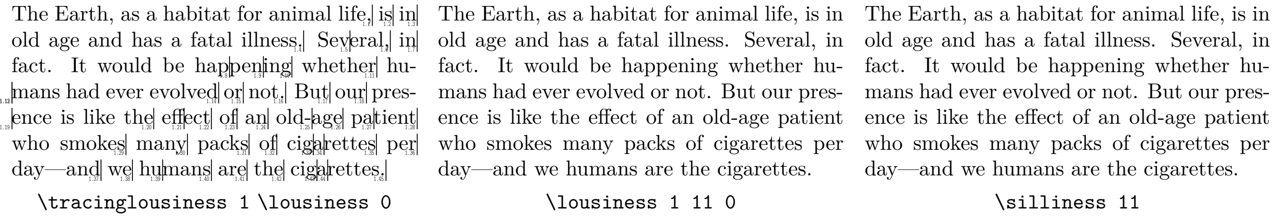
```
1   1   0  41684  veryloose  glue
2   2   1  51784  decent     glue
3   3   2  51980  decent     glue
4   4   3  52080  decent     glue
5   5   4  52336  decent     glue
6   6   5  65145  semiloose  glue
    7   5  65565  semitight  disc
7   8   6  75506  decent     glue
    9   7  88165  decent     disc
8  10   8  87027  semiloose  glue
   11   8  78490  decent     disc
9  12  10  88116  semiloose  glue
   13  11  78590  decent     glue

12   10 8 6 5 4 3 2 1
13   11 8 6 5 4 3 2 1
```

When we started playing with the par builder in the perspective of math, we side tracked and ended up with a feature that can ge used in controlled situations. Currently we only have a low level ConTEXt interface for this (see figure 2).

7 Criterion

The `granular` alignment option will configure the linebreakcriterion to work with 0.25 steps instead of 0.50 steps which means that successive lines can become a bit closer



The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

`\tracinglousiness 1 \lousiness 0`

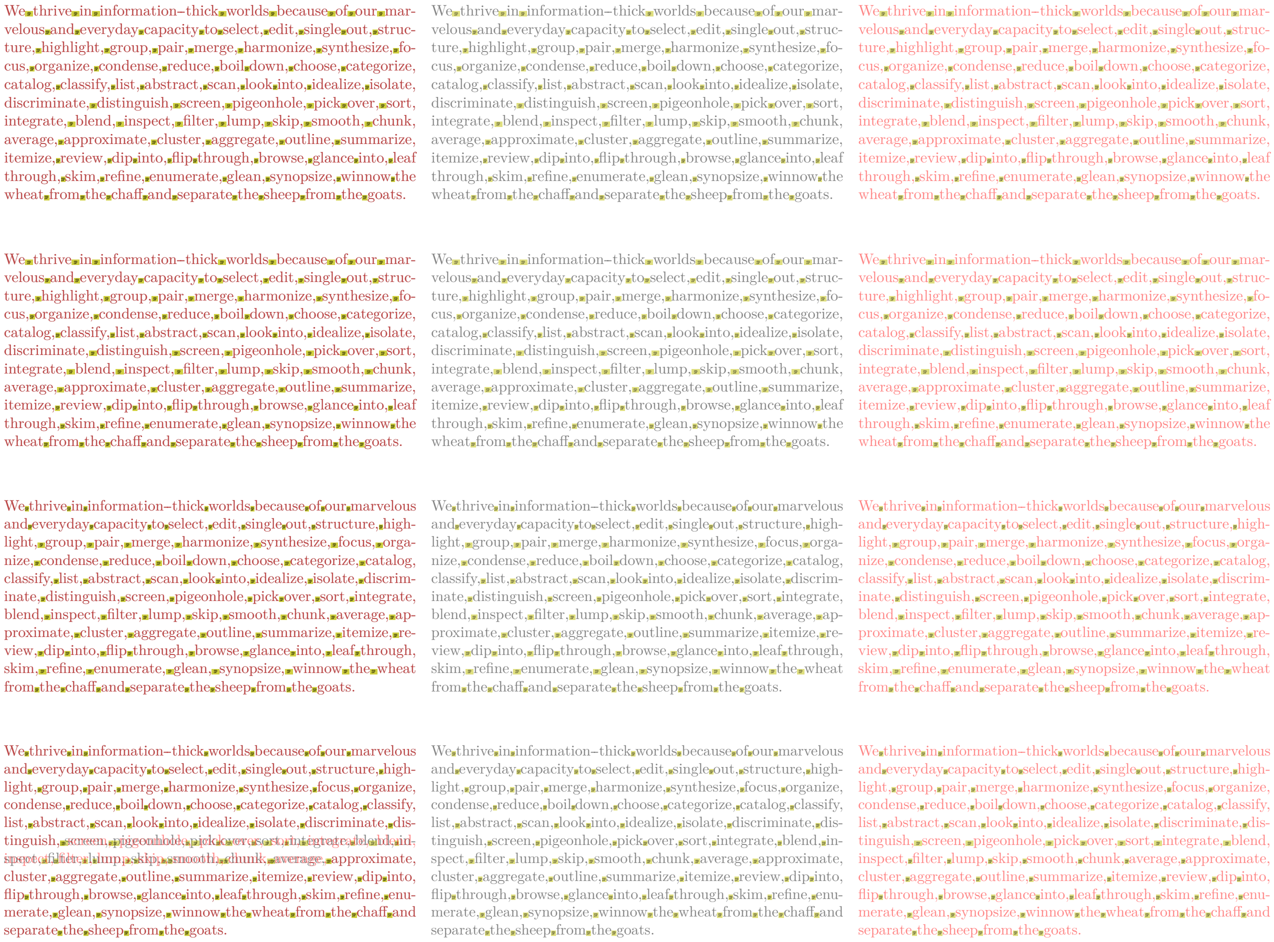
The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

`\lousiness 1 11 0`

The Earth, as a habitat for animal life, is in old age and has a fatal illness. Several, in fact. It would be happening whether humans had ever evolved or not. But our presence is like the effect of an old-age patient who smokes many packs of cigarettes per day—and we humans are the cigarettes.

`\silliness 11`

Figure 2 Influencing the way TEX breaks lines applied to `ward.tex`.



We thrive in information–thick worlds because of our marvelous and everyday capacity to select, edit, single out, structure, highlight, group, pair, merge, harmonize, synthesize, focus, organize, condense, reduce, boil down, choose, categorize, catalog, classify, list, abstract, scan, look into, idealize, isolate, discriminate, distinguish, screen, pigeonhole, pick over, sort, integrate, blend, inspect, filter, lump, skip, smooth, chunk, average, approximate, cluster, aggregate, outline, summarize, itemize, review, dip into, flip through, browse, glance into, leaf through, skim, refine, enumerate, glean, synopsize, winnow the wheat from the chaff and separate the sheep from the goats.

Figure 3 More granular interline criteria.

in spacing. There is no real impact on performance because testing happens anyway. In figure 3 you see some examples, where in some it indeed makes a difference.

8 Examples

The ConTEXt distribution comes with a few test setups: `spac-imp-tests.mkxl`*. Once we have found a suitable set of values and sample texts we might discuss them here.*

Currently we provide the following predefined passes that you can enable with `\setupalignpass`: `decent`, `quality`, `test1`, `test2`, `test3`, `test4`, `test5`. We hope that users are willing to test these.

9 Pages

While the par builder does multiple passes, the page builder is a single pass progressive routine. Every time something gets added to the (so called) main vertical list the page state gets updated and when the page overflows what has been collected gets passed to the output routine. It is to a large extend driven by glue (with stretch and shrink) and penalties and when content (boxes) is added the process is somewhat complicated by inserts as these needs to be taken into account too.

You can get pages that run from top to bottom by adding stretch between lines but by default in ConTEXt we prefer to fill up the bottom with white space.

It can be hard to make decisions at the TEX end around a potential page break because in order to get an idea how much space is left, one needs to trigger the page builder which can have side effects.

Penalties play an important role and because these are used to control for instance widows and clubs high values can lead to underfull pages so if we want to influence that we need to cheat. For this we have three experimental mechanisms:

- tweaking the page goal: `\pageextragoal`
- initializing the state quantities: `\initialpageskip`
- adapting the state quantities as we go: `\additionalpageskip`

The first tweak is for me to play with, and when a widow or club is seen the extra amount can kick in. This feature is likely to be replaced by a more configurable one.

The second tweak lets the empty page start out with some given height, stretch and shrink. This variable is persistent over pages. This is not true for the third tweak: it kicks in when the page gets initialized *or* as we go, but after it has been applied the value is reset. That makes it a feature like `\looseness`. We could combine these into one (because one can set up a persistent one in the macro package at well defined spots) but having an initial one also nicely can compensate the usual topskip glue hackery with a more natural control option.

Adapting the layout (within the regular text area) is done with `\setpagelooseness` an demonstrated in figure 4 and figure 5. Possible parameters are `lines`, `height`, `stretch` and `shrink`. You can also directly specify the number of lines. The other two features are not (yet) interfaced.

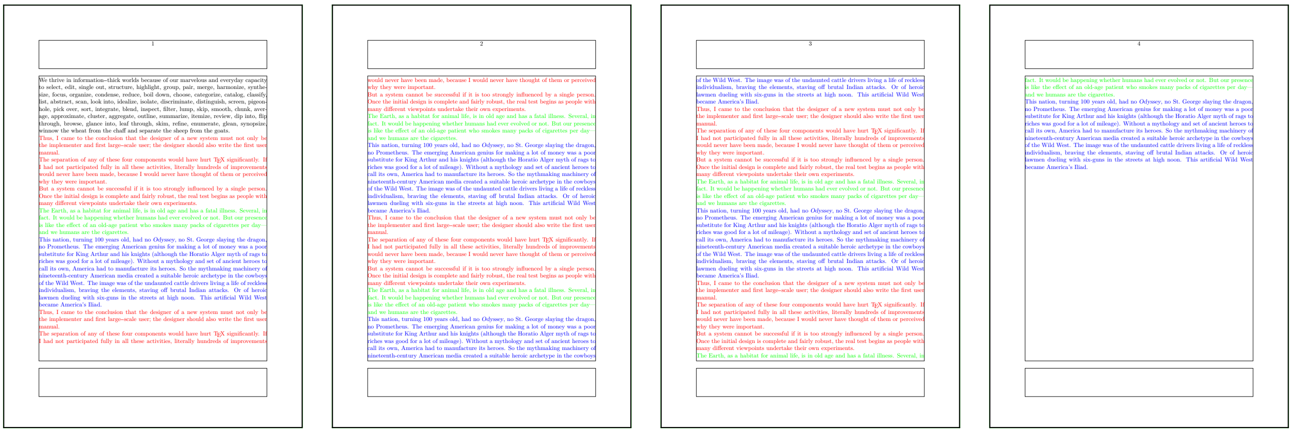


Figure 4 Cheating with page dimensions: `[lines=2]`.

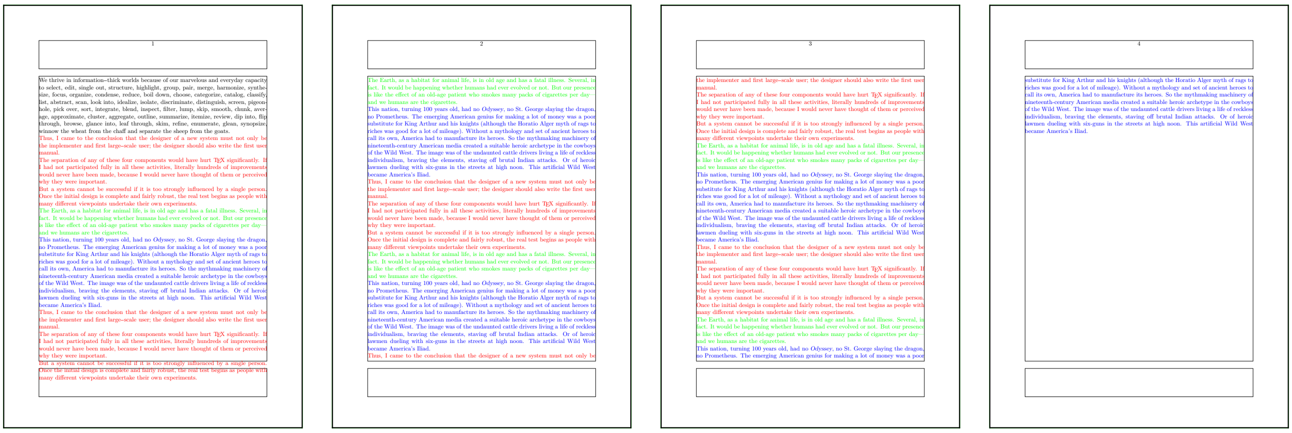


Figure 5 Cheating with page dimensions: `[-3]`.

It is not that trivial to fulfill the wide range of user demands but over time the `\setupalign` commands has gotten plenty of features. Getting for instance windows and clubs right in the kind of mixed usage that is common in ConTEXt is not always easy. One can experiment with scenarios (also to get some understanding of matters) but none is probably perfect (unless one does something close to manual tweaking). There is also the butterfly effect: a change here might trigger na issue there.

The examples in figure 6, 7 and 8 scale vertically in order ti fill up the text area; the `vz` parameter is set with `setuplayout`. In the example the widow and club penalties are set to 10000. In these examples we have enabled the `layout.vz` trackers that shows a small black rule indicating the amount of stretch.

There are a few other tweaks but these one can wonder about these. We can add stretch and shrink to the baseline skip, something that can also be triggered with the ‘spread’ option to `\setupalign`, assuming that also `height` is given). An alternative is to permit an extra line and accept a visual overflow, assuming that the layout is set up to make sure that the footer line doesn't overlap. None of this guarantees that a whole document

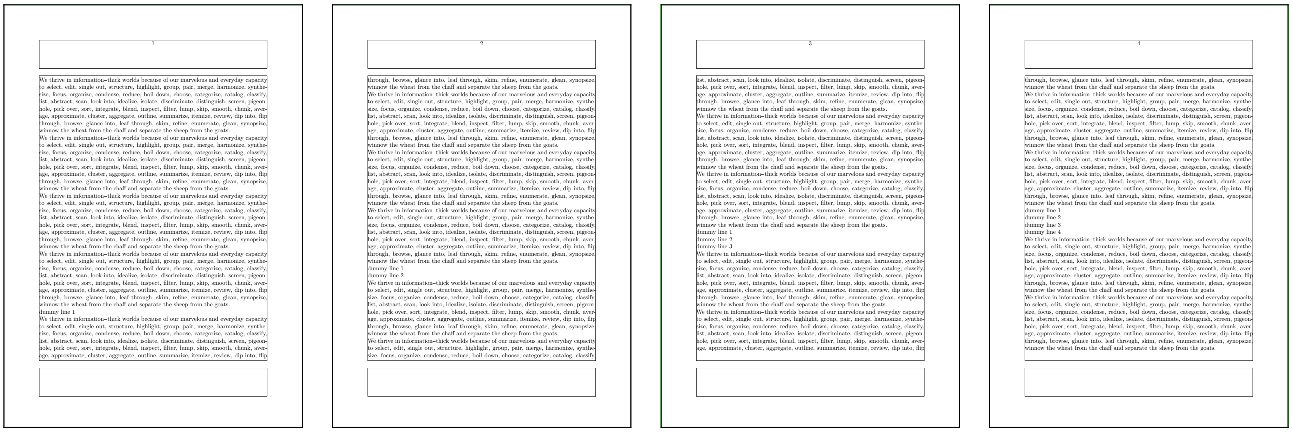


Figure 6 Cheating with vertical expansion: `[vz=no]`.

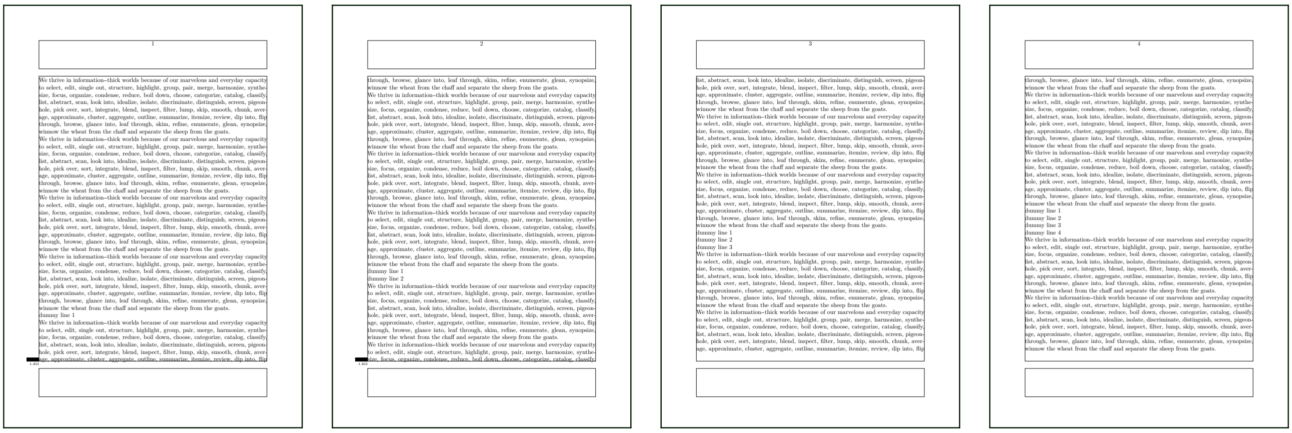


Figure 7 Cheating with vertical expansion: `[vz=yes]`.

with plenty of graphics and special constructs will come out well, but for text only it might work okay. Figures 9, 10 and 11 show some of this.

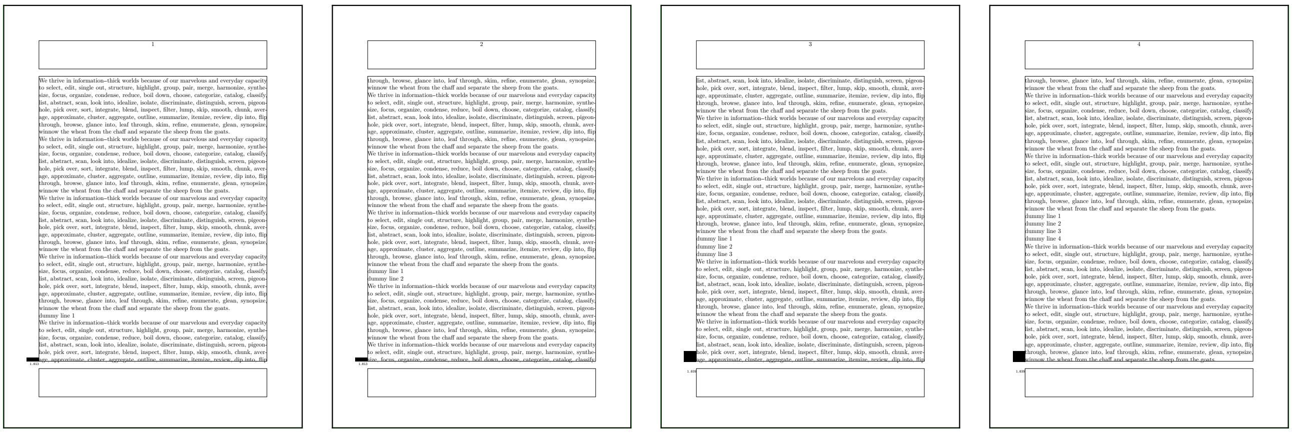


Figure 8 Cheating with vertical expansion: `[vz=2]`.

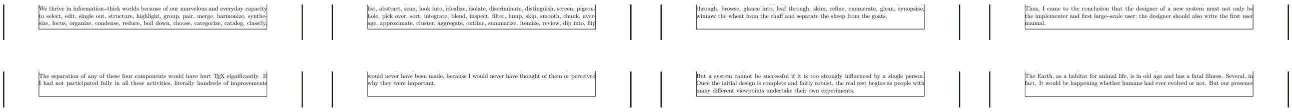


Figure 9 Cheating: just high penalties.

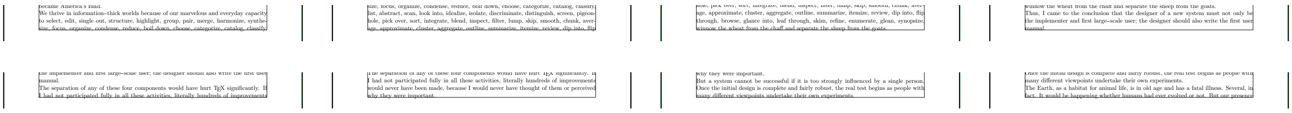


Figure 10 Cheating: `\baselineskip 1\baselineskip plus 1pt minus .1pt`.

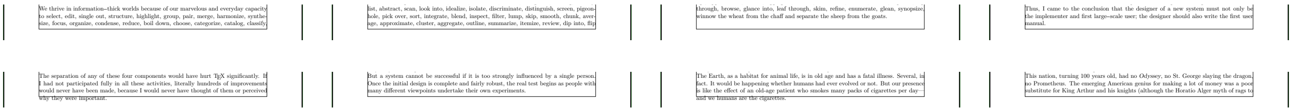


Figure 11 Cheating: `\pageextragoal\lineheight`.

10 Profiles

You can have a paragraph with lines that exceed the maximum height and/or depth or where spaces end up in a way that create so called rivers. Rivers are more a curiosity than an annoyance because any attempt to avoid them is likely to result in a worse looking result. The unequal line distances can be annoying too but these can be avoided when bringing lines closer together doesn't lead to clashes. This can be done without reformatting the paragraph by passing the `profile` option to `\setupalign`. It comes at the cost of a little more runtime and (as far as we observed) it kicks in seldom, for instance when inline math is used that has super- or subscripts, radicals, fractions or other slightly higher constructs.

10 Colofon

Author	Hans Hagen & Mikael Sundqvist
ConTEXt	2023.08.10 02:59
LuaMetaTEX	211.0
Support	www.pragma-ade.com contextgarden.net